**Steps to implement Spring security**

Contents

[**I.** **Spring security installation** 2](#_Toc92297176)

[1. Install spring security and jwt library 2](#_Toc92297177)

[2. Tạo model User để quản lý user của hệ thống 3](#_Toc92297178)

[3. Tạo repository “AppUserRepository.java” thao tác với table user 5](#_Toc92297179)

[4. Tạo service “IAppUserService.java” và “AppUserServiceImpl.java” thao xử lý logic với table user 6](#_Toc92297180)

[II. **Config spring security to apply authenticated and authorization** 10](#_Toc92297181)

[1. Create “JwtTokenUtil.Java” This class contains all useful method to process with token 10](#_Toc92297182)

[2. Implement “SessionUser.java” 12](#_Toc92297183)

[3. Create “JwtUserDetailsService.java” to implement UserDetailsService 14](#_Toc92297184)

[4. Create “JwtAuthenticationEntryPoint.java” to handle 401 Unauthorize 15](#_Toc92297185)

[5. Add “JwtRequestFilter.java” to filter all coming request 16](#_Toc92297186)

[6. Add Utils “SessionHelper.java” 18](#_Toc92297187)

[7. Add request and response payload structure 19](#_Toc92297188)

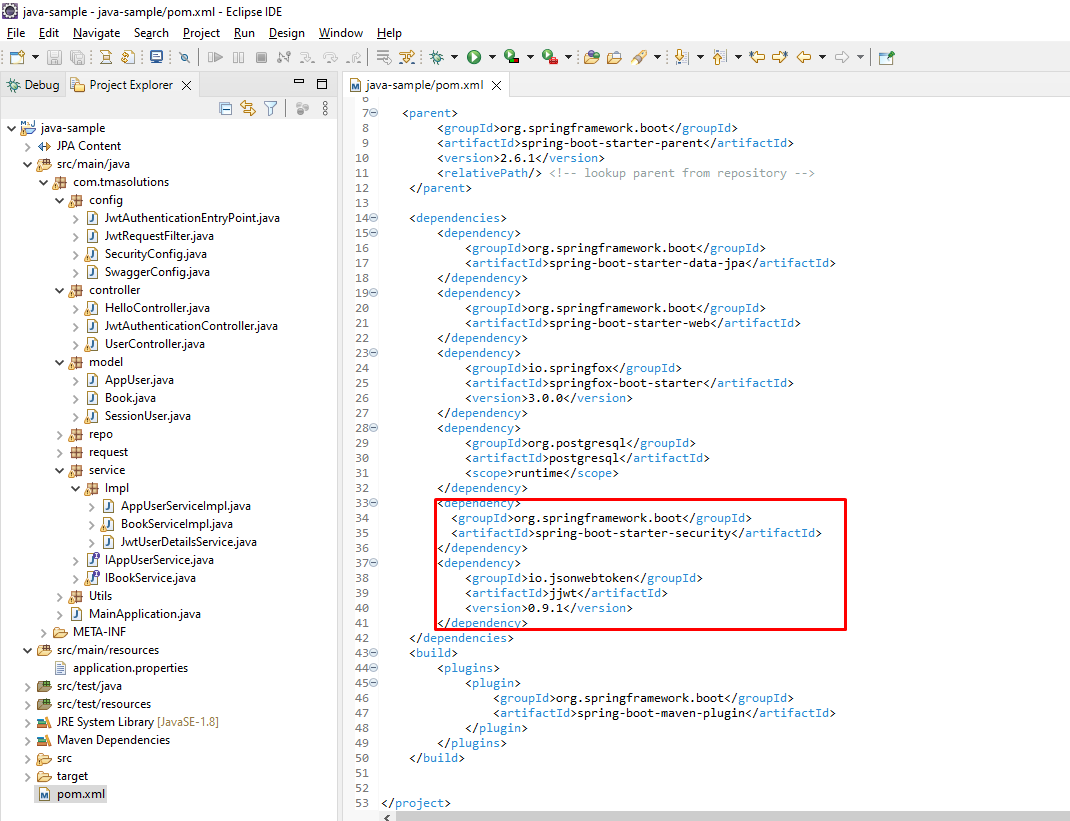
[8. Create “SecurityConfig.java” to manage all spring security configuration 20](#_Toc92297189)

[9. Create controller “JwtAuthenticationController” to login and return jwt token 22](#_Toc92297190)

[**III.** **Modify swagger UI to apply authorization to swagger** 24](#_Toc92297191)

[1. Modified swagger ui to allow authorize 24](#_Toc92297192)

1. **Spring security installation**
2. Install spring security and jwt library



<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

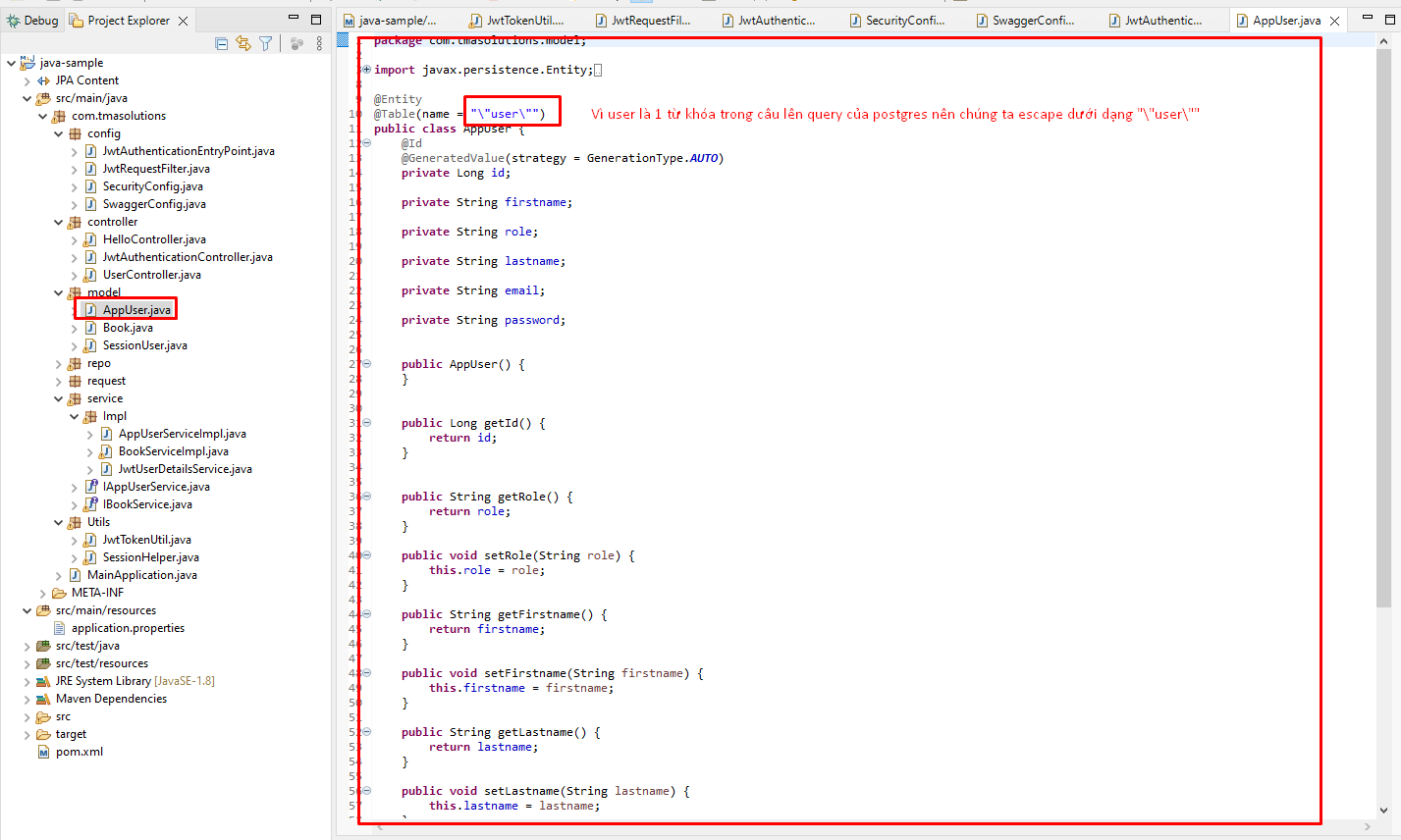
<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

1. Tạo model User để quản lý user của hệ thống

(Lưu ý: vì trong spring security có native class User rồi nên để tránh conflict class name ta có thể đặt class của mình khác “User.java”)



**package** com.tmasolutions.model;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "\"user\"")

**public** **class** AppUser {

@Id

@GeneratedValue(strategy = GenerationType.***AUTO***)

**private** Long id;

**private** String firstname;

**private** String role;

**private** String lastname;

**private** String email;

**private** String password;

**public** AppUser() {

}

**public** Long getId() {

**return** id;

}

**public** String getRole() {

**return** role;

}

**public** **void** setRole(String role) {

**this**.role = role;

}

**public** String getFirstname() {

**return** firstname;

}

**public** **void** setFirstname(String firstname) {

**this**.firstname = firstname;

}

**public** String getLastname() {

**return** lastname;

}

**public** **void** setLastname(String lastname) {

**this**.lastname = lastname;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getPassword() {

**return** password;

}

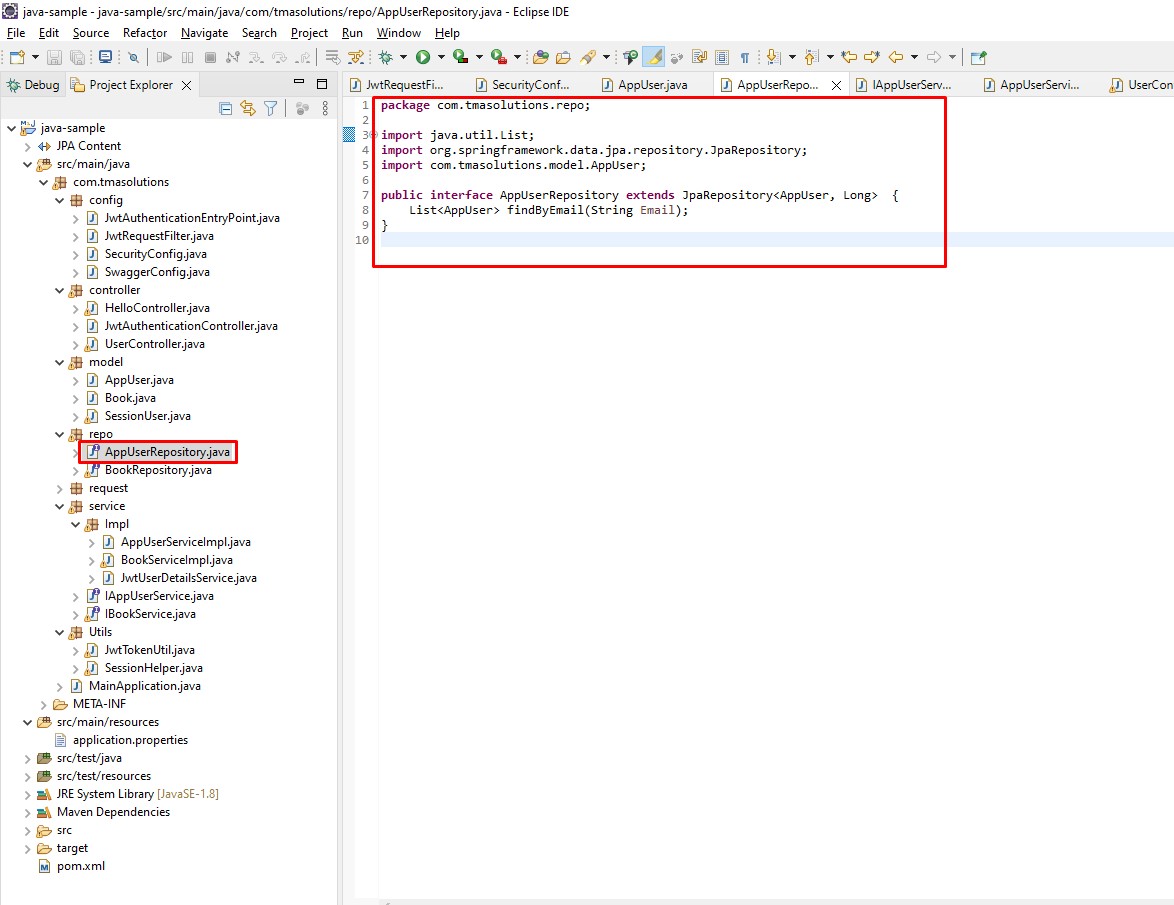
**public** **void** setPassword(String password) {

**this**.password = password;

}

}

1. Tạo repository “AppUserRepository.java” thao tác với table user



**package** com.tmasolutions.repo;

**import** java.util.List;

**import** org.springframework.data.jpa.repository.JpaRepository;

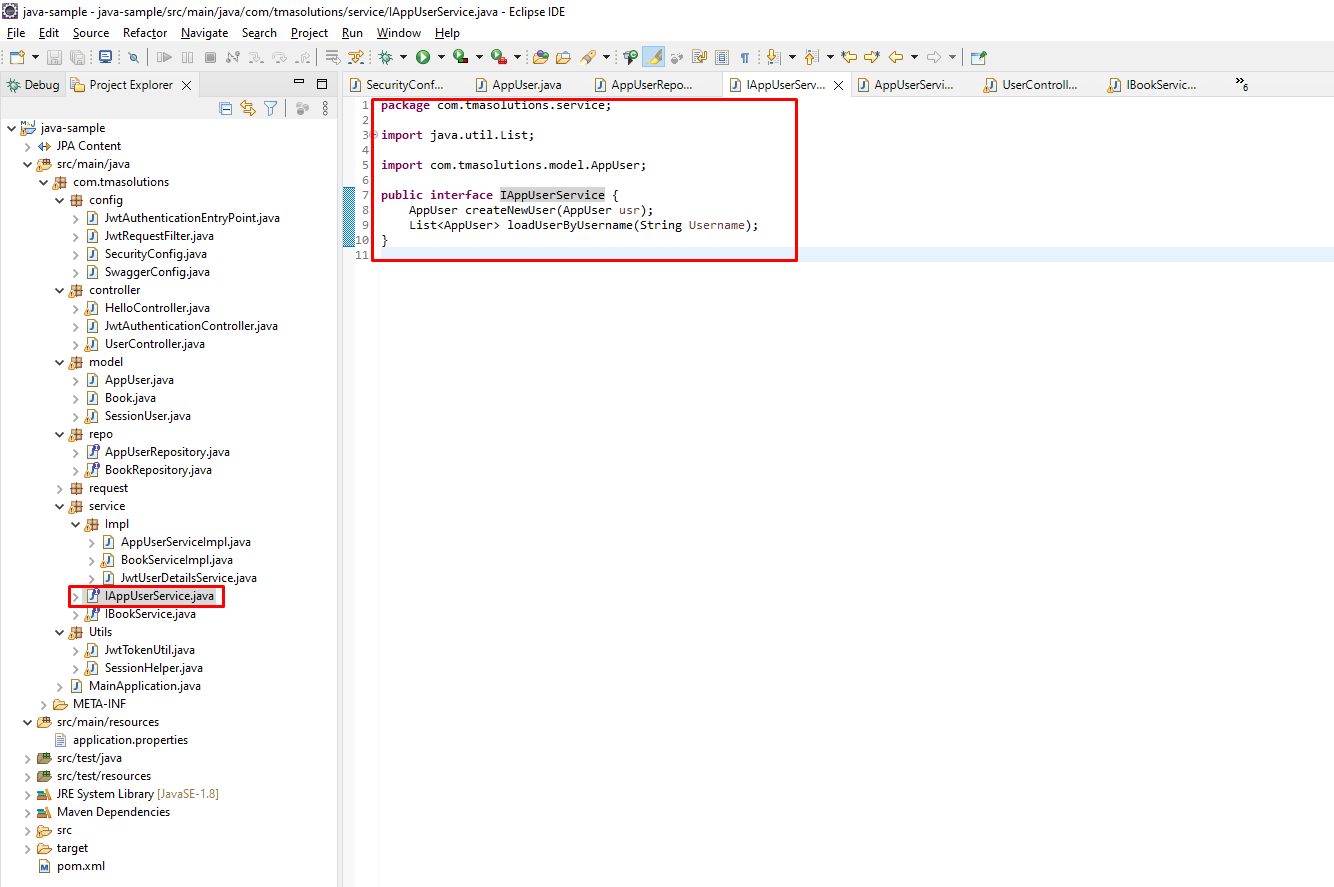
**import** com.tmasolutions.model.AppUser;

**public** **interface** AppUserRepository **extends** JpaRepository<AppUser, Long> {

List<AppUser> findByEmail(String Email);

}

1. Tạo service “IAppUserService.java” và “AppUserServiceImpl.java” thao xử lý logic với table user



**package** com.tmasolutions.service;

**import** java.util.List;

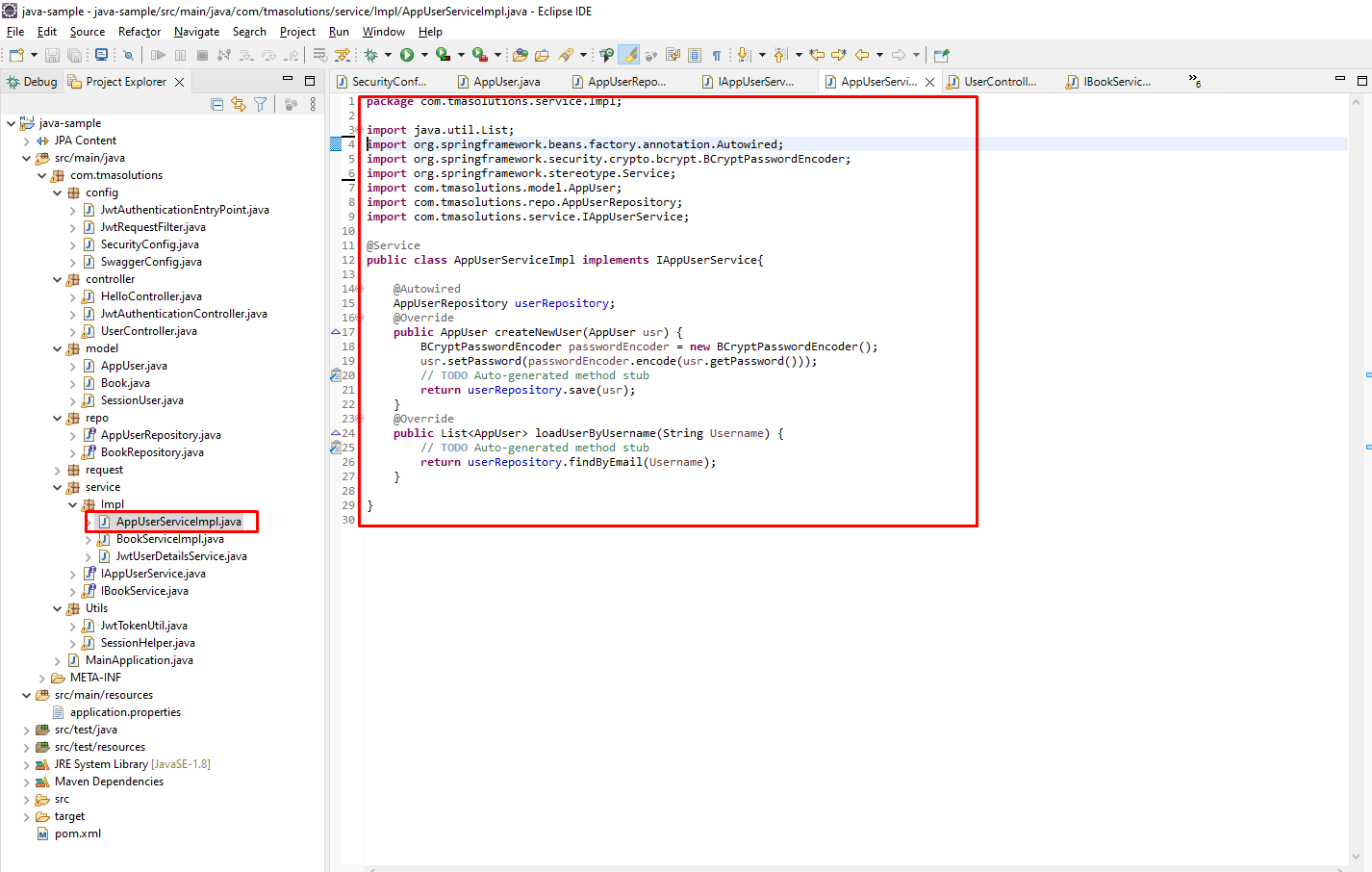
**import** com.tmasolutions.model.AppUser;

**public** **interface** IAppUserService {

AppUser createNewUser(AppUser usr);

List<AppUser> loadUserByUsername(String Username);

}



**package** com.tmasolutions.service.Impl;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

**import** org.springframework.stereotype.Service;

**import** com.tmasolutions.model.AppUser;

**import** com.tmasolutions.repo.AppUserRepository;

**import** com.tmasolutions.service.IAppUserService;

@Service

**public** **class** AppUserServiceImpl **implements** IAppUserService{

@Autowired

AppUserRepository userRepository;

@Override

**public** AppUser createNewUser(AppUser usr) {

BCryptPasswordEncoder passwordEncoder = **new** BCryptPasswordEncoder();

usr.setPassword(passwordEncoder.encode(usr.getPassword()));

// **TODO** Auto-generated method stub

**return** userRepository.save(usr);

}

@Override

**public** List<AppUser> loadUserByUsername(String Username) {

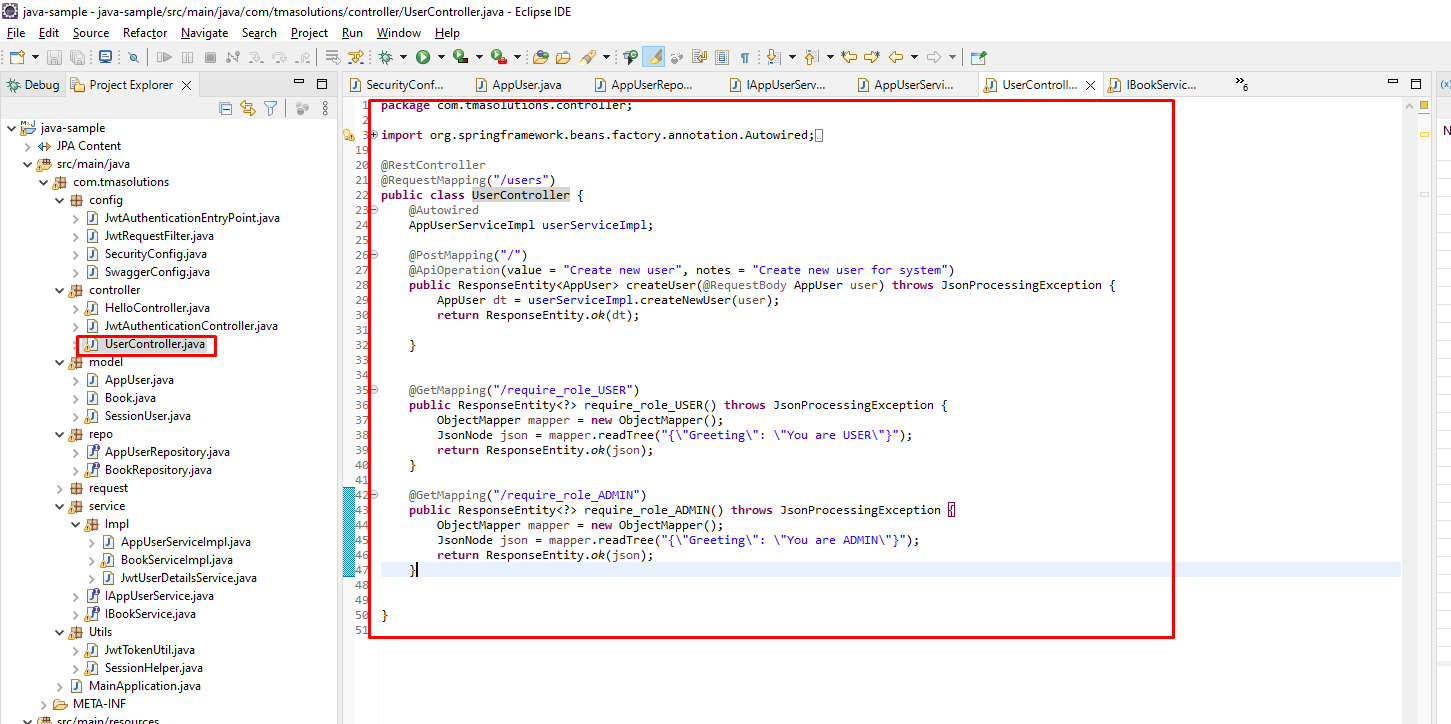
// **TODO** Auto-generated method stub

**return** userRepository.findByEmail(Username);

}

}

1. Tạo controller “UserController.java” để quản lý API thêm/ xóa/ sửa user



**package** com.tmasolutions.controller;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** com.fasterxml.jackson.core.JsonProcessingException;

**import** com.fasterxml.jackson.databind.JsonNode;

**import** com.fasterxml.jackson.databind.ObjectMapper;

**import** com.tmasolutions.model.AppUser;

**import** com.tmasolutions.service.Impl.AppUserServiceImpl;

**import** io.swagger.annotations.ApiOperation;

@RestController

@RequestMapping("/users")

**public** **class** UserController {

@Autowired

AppUserServiceImpl userServiceImpl;

@PostMapping("/")

@ApiOperation(value = "Create new user", notes = "Create new user for system")

**public** ResponseEntity<AppUser> createUser(@RequestBody AppUser user) **throws** JsonProcessingException {

AppUser dt = userServiceImpl.createNewUser(user);

**return** ResponseEntity.*ok*(dt);

}

@GetMapping("/require\_role\_USER")

**public** ResponseEntity<?> require\_role\_USER() **throws** JsonProcessingException {

ObjectMapper mapper = **new** ObjectMapper();

JsonNode json = mapper.readTree("{\"Greeting\": \"You are USER\"}");

**return** ResponseEntity.*ok*(json);

}

@GetMapping("/require\_role\_ADMIN")

**public** ResponseEntity<?> require\_role\_ADMIN() **throws** JsonProcessingException {

ObjectMapper mapper = **new** ObjectMapper();

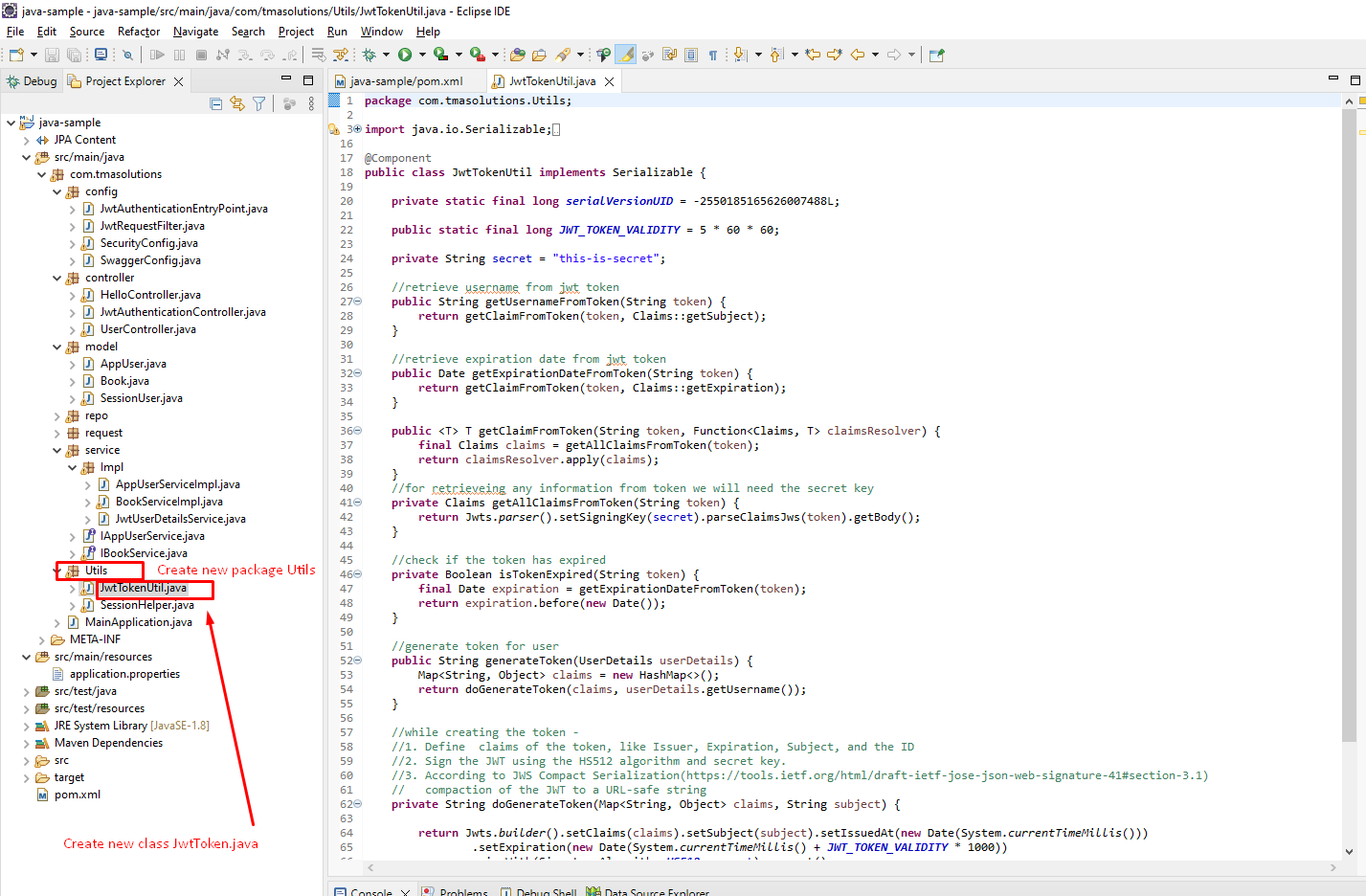
JsonNode json = mapper.readTree("{\"Greeting\": \"You are ADMIN\"}");

**return** ResponseEntity.*ok*(json);

}

}

1. **Config spring security to apply authenticated and authorization**
2. Create “JwtTokenUtil.Java” This class contains all useful method to process with token



package com.tmasolutions.Utils;

import java.io.Serializable;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

@Component

public class JwtTokenUtil implements Serializable {

private static final long serialVersionUID = -2550185165626007488L;

public static final long JWT\_TOKEN\_VALIDITY = 5 \* 60 \* 60;

private String secret = "this-is-secret";

//retrieve username from jwt token

public String getUsernameFromToken(String token) {

return getClaimFromToken(token, Claims::getSubject);

}

//retrieve expiration date from jwt token

public Date getExpirationDateFromToken(String token) {

return getClaimFromToken(token, Claims::getExpiration);

}

public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {

final Claims claims = getAllClaimsFromToken(token);

return claimsResolver.apply(claims);

}

//for retrieveing any information from token we will need the secret key

private Claims getAllClaimsFromToken(String token) {

return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();

}

//check if the token has expired

private Boolean isTokenExpired(String token) {

final Date expiration = getExpirationDateFromToken(token);

return expiration.before(new Date());

}

//generate token for user

public String generateToken(UserDetails userDetails) {

Map<String, Object> claims = new HashMap<>();

return doGenerateToken(claims, userDetails.getUsername());

}

//while creating the token -

//1. Define claims of the token, like Issuer, Expiration, Subject, and the ID

//2. Sign the JWT using the HS512 algorithm and secret key.

//3. According to JWS Compact Serialization(https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-41#section-3.1)

// compaction of the JWT to a URL-safe string

private String doGenerateToken(Map<String, Object> claims, String subject) {

return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + JWT\_TOKEN\_VALIDITY \* 1000))

.signWith(SignatureAlgorithm.HS512, secret).compact();

}

//validate token

public Boolean validateToken(String token, UserDetails userDetails) {

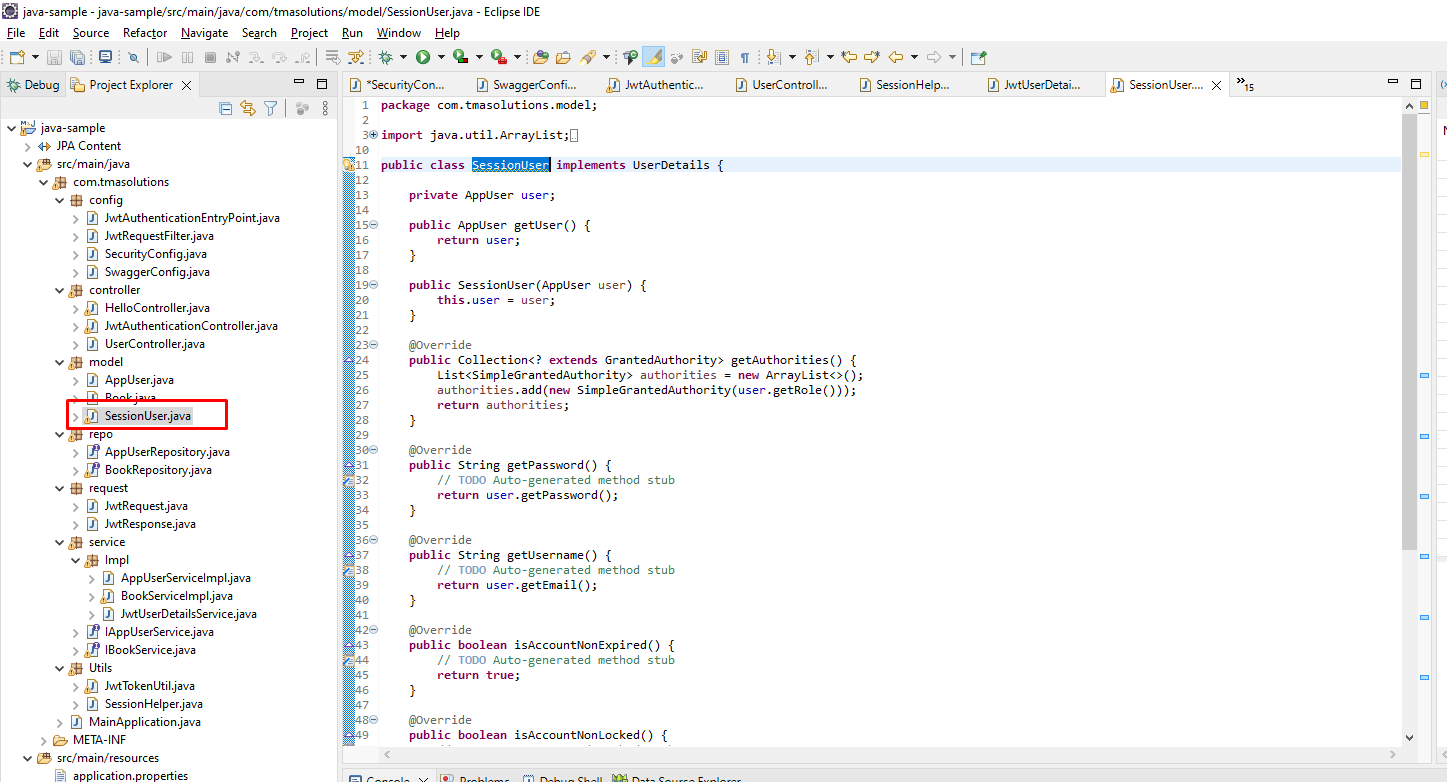
final String username = getUsernameFromToken(token);

return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));

}

}

1. Implement “SessionUser.java”



1. Create “JwtUserDetailsService.java” to implement UserDetailsService

package com.tmasolutions.model;

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.security.core.userdetails.UserDetails;

public class SessionUser implements UserDetails {

private AppUser user;

public AppUser getUser() {

return user;

}

public SessionUser(AppUser user) {

this.user = user;

}

@Override

public Collection<? extends GrantedAuthority> getAuthorities() {

List<SimpleGrantedAuthority> authorities = new ArrayList<>();

authorities.add(new SimpleGrantedAuthority(user.getRole()));

return authorities;

}

@Override

public String getPassword() {

// TODO Auto-generated method stub

return user.getPassword();

}

@Override

public String getUsername() {

// TODO Auto-generated method stub

return user.getEmail();

}

@Override

public boolean isAccountNonExpired() {

// TODO Auto-generated method stub

return true;

}

@Override

public boolean isAccountNonLocked() {

// TODO Auto-generated method stub

return true;

}

@Override

public boolean isCredentialsNonExpired() {

// TODO Auto-generated method stub

return true;

}

@Override

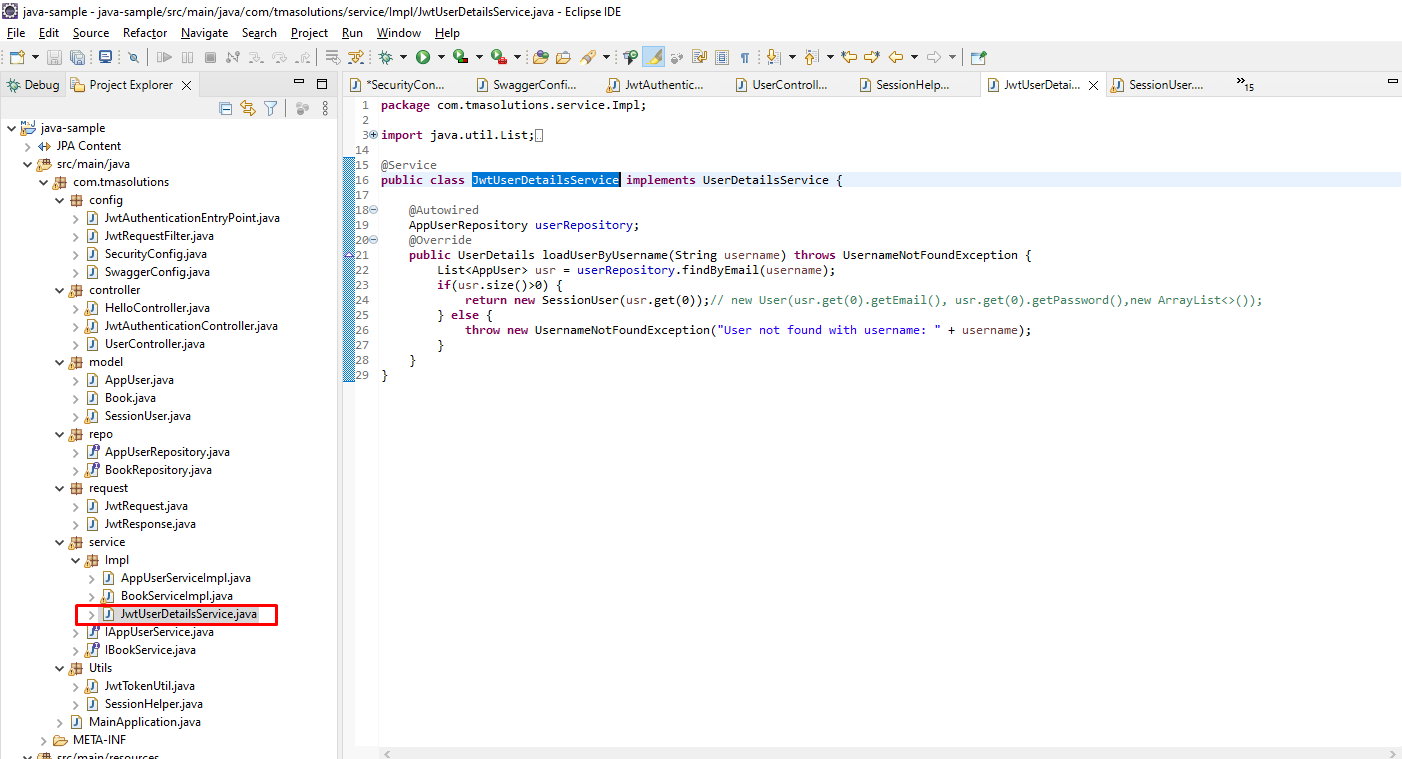
public boolean isEnabled() {

// TODO Auto-generated method stub

return true;

}

}



package com.tmasolutions.service.Impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Service;

import com.tmasolutions.model.AppUser;

import com.tmasolutions.model.SessionUser;

import com.tmasolutions.repo.AppUserRepository;

@Service

public class JwtUserDetailsService implements UserDetailsService {

@Autowired

AppUserRepository userRepository;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

List<AppUser> usr = userRepository.findByEmail(username);

if(usr.size()>0) {

return new SessionUser(usr.get(0));// new User(usr.get(0).getEmail(), usr.get(0).getPassword(),new ArrayList<>());

} else {

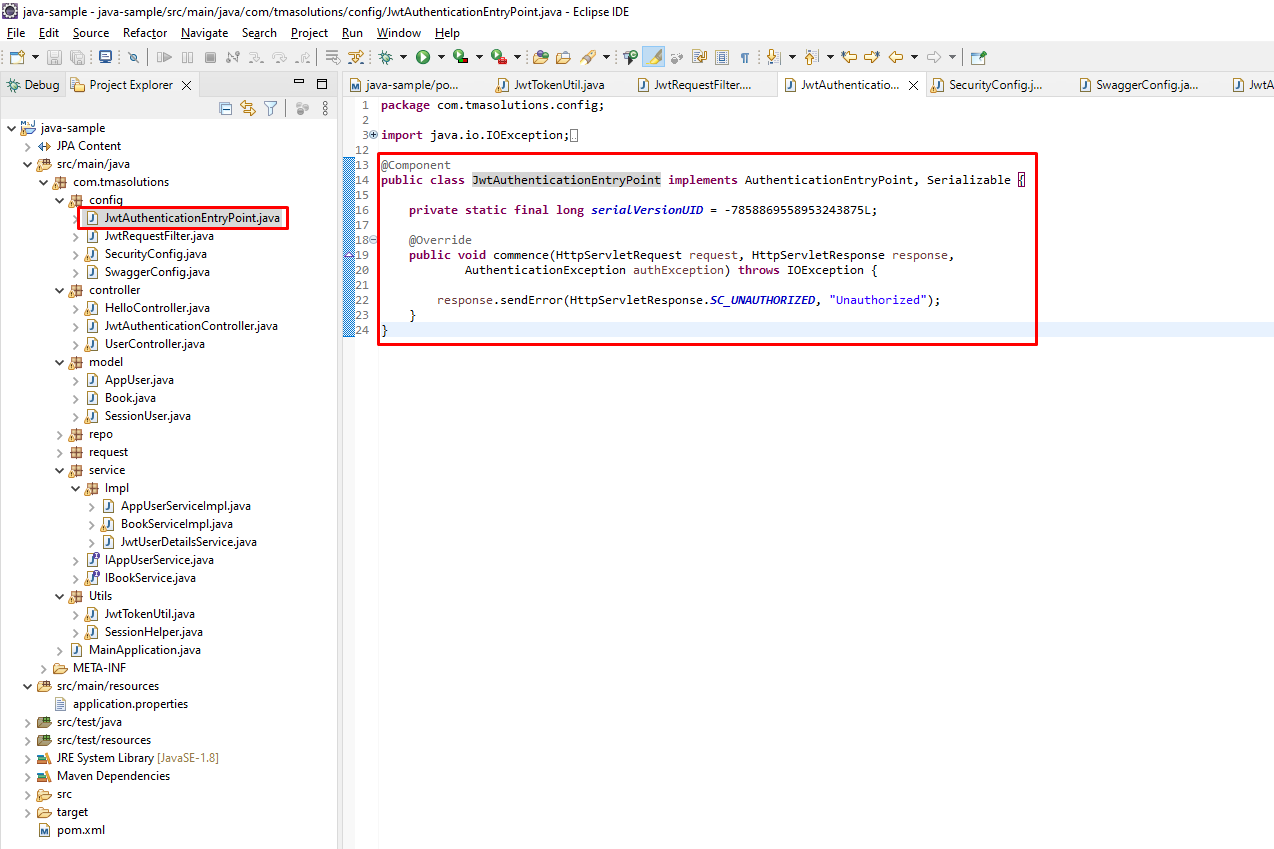
throw new UsernameNotFoundException("User not found with username: " + username);

}

}

}

1. Create “JwtAuthenticationEntryPoint.java” to handle 401 Unauthorize



package com.tmasolutions.config;

import java.io.IOException;

import java.io.Serializable;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.AuthenticationException;

import org.springframework.security.web.AuthenticationEntryPoint;

import org.springframework.stereotype.Component;

@Component

public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint, Serializable {

private static final long serialVersionUID = -7858869558953243875L;

@Override

public void commence(HttpServletRequest request, HttpServletResponse response,

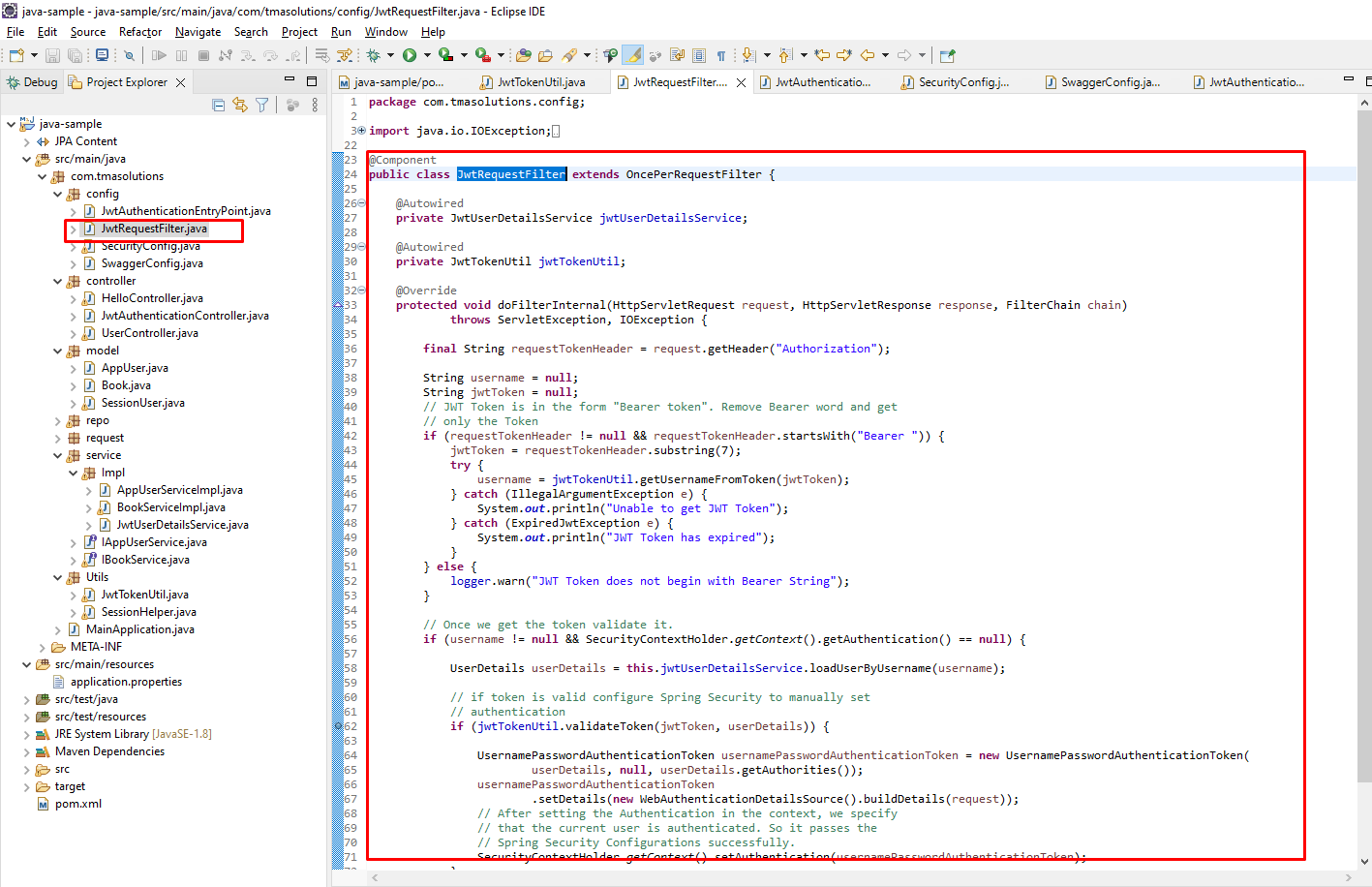
AuthenticationException authException) throws IOException {

response.sendError(HttpServletResponse.SC\_UNAUTHORIZED, "Unauthorized");

}

}

1. Add “JwtRequestFilter.java” to filter all coming request



package com.tmasolutions.config;

import java.io.IOException;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import com.tmasolutions.Utils.JwtTokenUtil;

import com.tmasolutions.service.Impl.JwtUserDetailsService;

import io.jsonwebtoken.ExpiredJwtException;

@Component

public class JwtRequestFilter extends OncePerRequestFilter {

@Autowired

private JwtUserDetailsService jwtUserDetailsService;

@Autowired

private JwtTokenUtil jwtTokenUtil;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

throws ServletException, IOException {

final String requestTokenHeader = request.getHeader("Authorization");

String username = null;

String jwtToken = null;

// JWT Token is in the form "Bearer token". Remove Bearer word and get

// only the Token

if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {

jwtToken = requestTokenHeader.substring(7);

try {

username = jwtTokenUtil.getUsernameFromToken(jwtToken);

} catch (IllegalArgumentException e) {

System.out.println("Unable to get JWT Token");

} catch (ExpiredJwtException e) {

System.out.println("JWT Token has expired");

}

} else {

logger.warn("JWT Token does not begin with Bearer String");

}

// Once we get the token validate it.

if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

UserDetails userDetails = this.jwtUserDetailsService.loadUserByUsername(username);

// if token is valid configure Spring Security to manually set

// authentication

if (jwtTokenUtil.validateToken(jwtToken, userDetails)) {

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(

userDetails, null, userDetails.getAuthorities());

usernamePasswordAuthenticationToken

.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

// After setting the Authentication in the context, we specify

// that the current user is authenticated. So it passes the

// Spring Security Configurations successfully.

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);

}

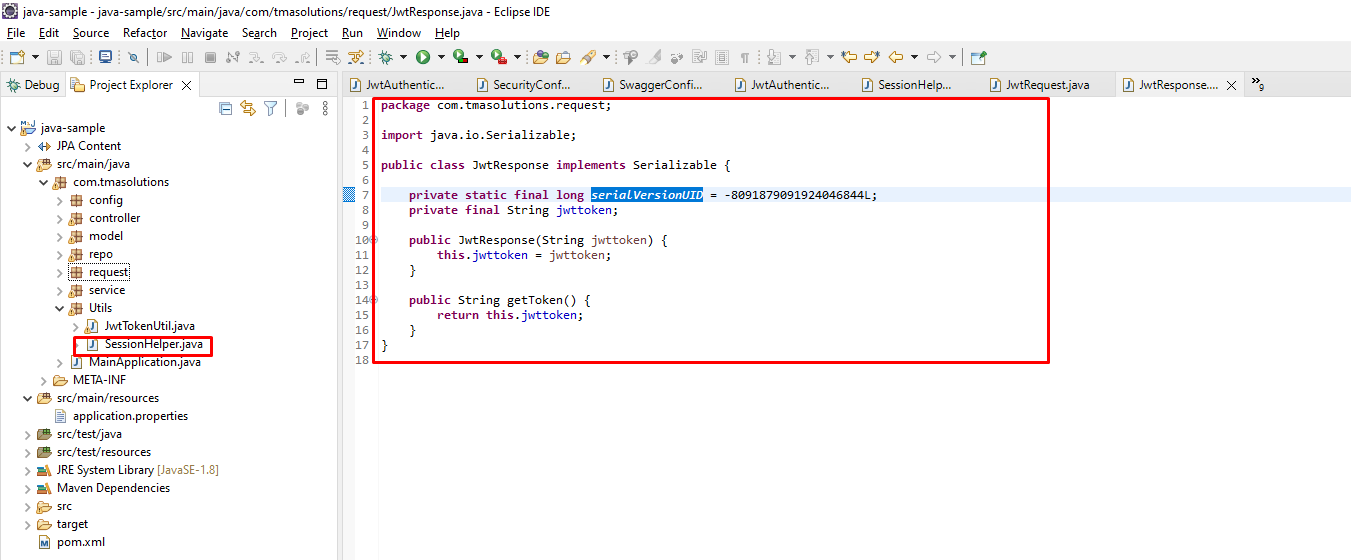
}

chain.doFilter(request, response);

}

}

1. Add Utils “SessionHelper.java”



**package** com.tmasolutions.Utils;

**import** org.springframework.security.authentication.AnonymousAuthenticationToken;

**import** org.springframework.security.core.Authentication;

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** com.tmasolutions.model.AppUser;

**import** com.tmasolutions.model.SessionUser;

**public** **class** SessionHelper {

**public** **static** AppUser getCurrentUser() {

Authentication authentication = SecurityContextHolder.*getContext*().getAuthentication();

**if** (!(authentication **instanceof** AnonymousAuthenticationToken)) {

**return** ((SessionUser)authentication.getPrincipal()).getUser();

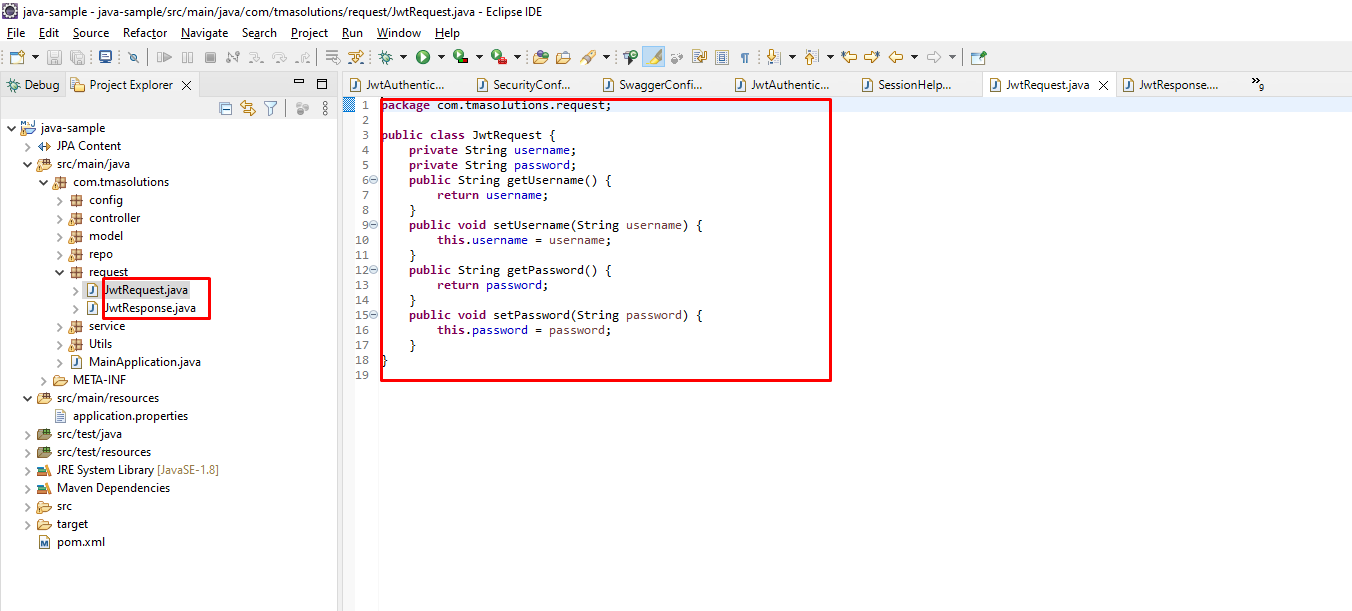
}

**return** **null**;

}

}

1. Add request and response payload structure



**package** com.tmasolutions.request;

**public** **class** JwtRequest {

**private** String username;

**private** String password;

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

}

**package** com.tmasolutions.request;

**import** java.io.Serializable;

**public** **class** JwtResponse **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = -8091879091924046844L;

**private** **final** String jwttoken;

**public** JwtResponse(String jwttoken) {

**this**.jwttoken = jwttoken;

}

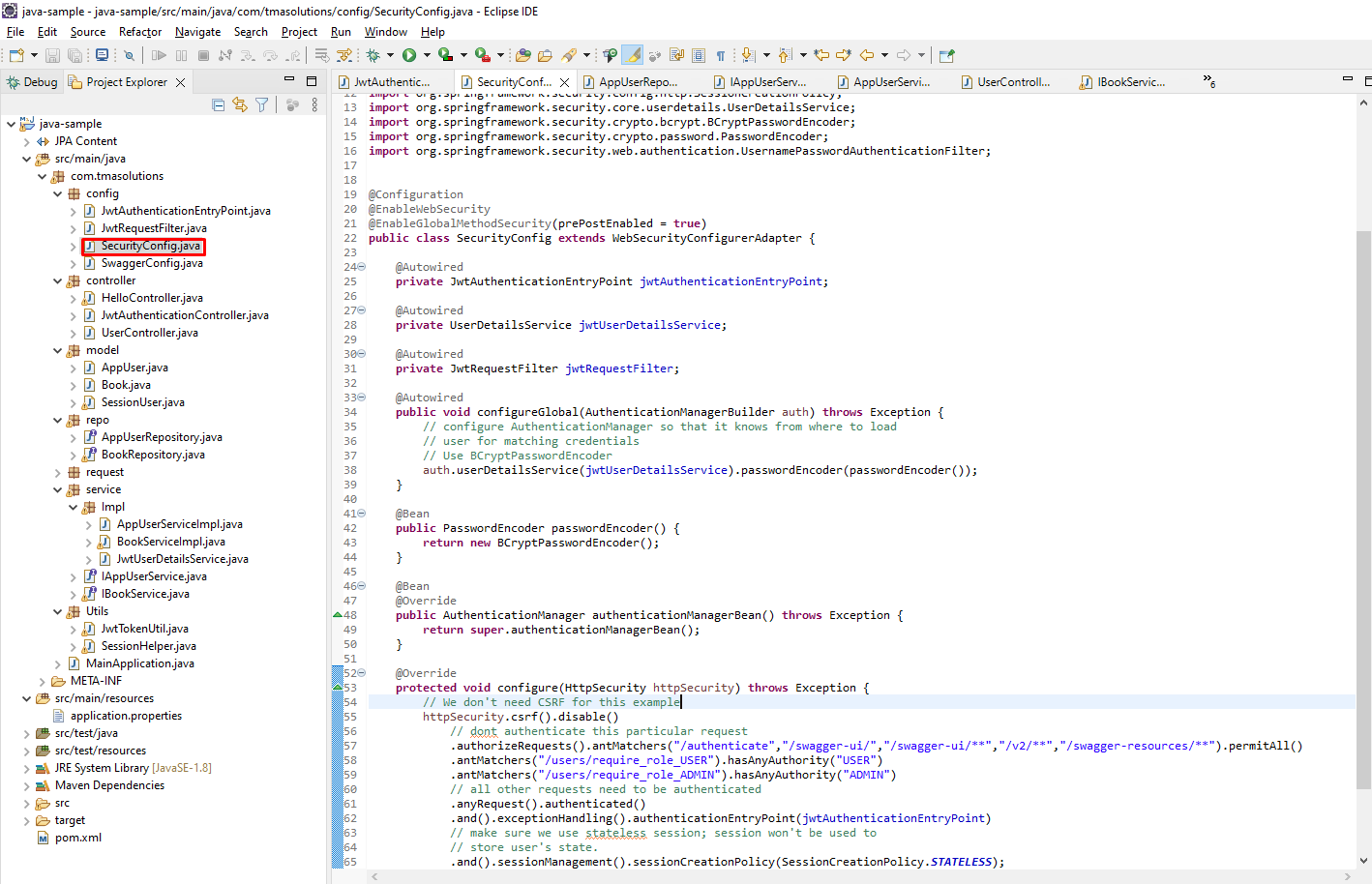
**public** String getToken() {

**return** **this**.jwttoken;

}

}

1. Create “SecurityConfig.java” to manage all spring security configuration



**package** com.tmasolutions.config;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.security.authentication.AuthenticationManager;

**import** org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

**import** org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

**import** org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

**import** org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

**import** org.springframework.security.config.http.SessionCreationPolicy;

**import** org.springframework.security.core.userdetails.UserDetailsService;

**import** org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

**import** org.springframework.security.crypto.password.PasswordEncoder;

**import** org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration

@EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = **true**)

**public** **class** SecurityConfig **extends** WebSecurityConfigurerAdapter {

@Autowired

**private** JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

@Autowired

**private** UserDetailsService jwtUserDetailsService;

@Autowired

**private** JwtRequestFilter jwtRequestFilter;

@Autowired

**public** **void** configureGlobal(AuthenticationManagerBuilder auth) **throws** Exception {

// configure AuthenticationManager so that it knows from where to load

// user for matching credentials

// Use BCryptPasswordEncoder

auth.userDetailsService(jwtUserDetailsService).passwordEncoder(passwordEncoder());

}

@Bean

**public** PasswordEncoder passwordEncoder() {

**return** **new** BCryptPasswordEncoder();

}

@Bean

@Override

**public** AuthenticationManager authenticationManagerBean() **throws** Exception {

**return** **super**.authenticationManagerBean();

}

@Override

**protected** **void** configure(HttpSecurity httpSecurity) **throws** Exception {

// We don't need CSRF for this example

httpSecurity.csrf().disable()

// dont authenticate this particular request

.authorizeRequests().antMatchers("/authenticate","/swagger-ui/","/swagger-ui/\*\*","/v2/\*\*","/swagger-resources/\*\*").permitAll()

.antMatchers("/users/require\_role\_USER").hasAnyAuthority("USER")

.antMatchers("/users/require\_role\_ADMIN").hasAnyAuthority("ADMIN")

// all other requests need to be authenticated

.anyRequest().authenticated()

.and().exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint)

// make sure we use stateless session; session won't be used to

// store user's state.

.and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.***STATELESS***);

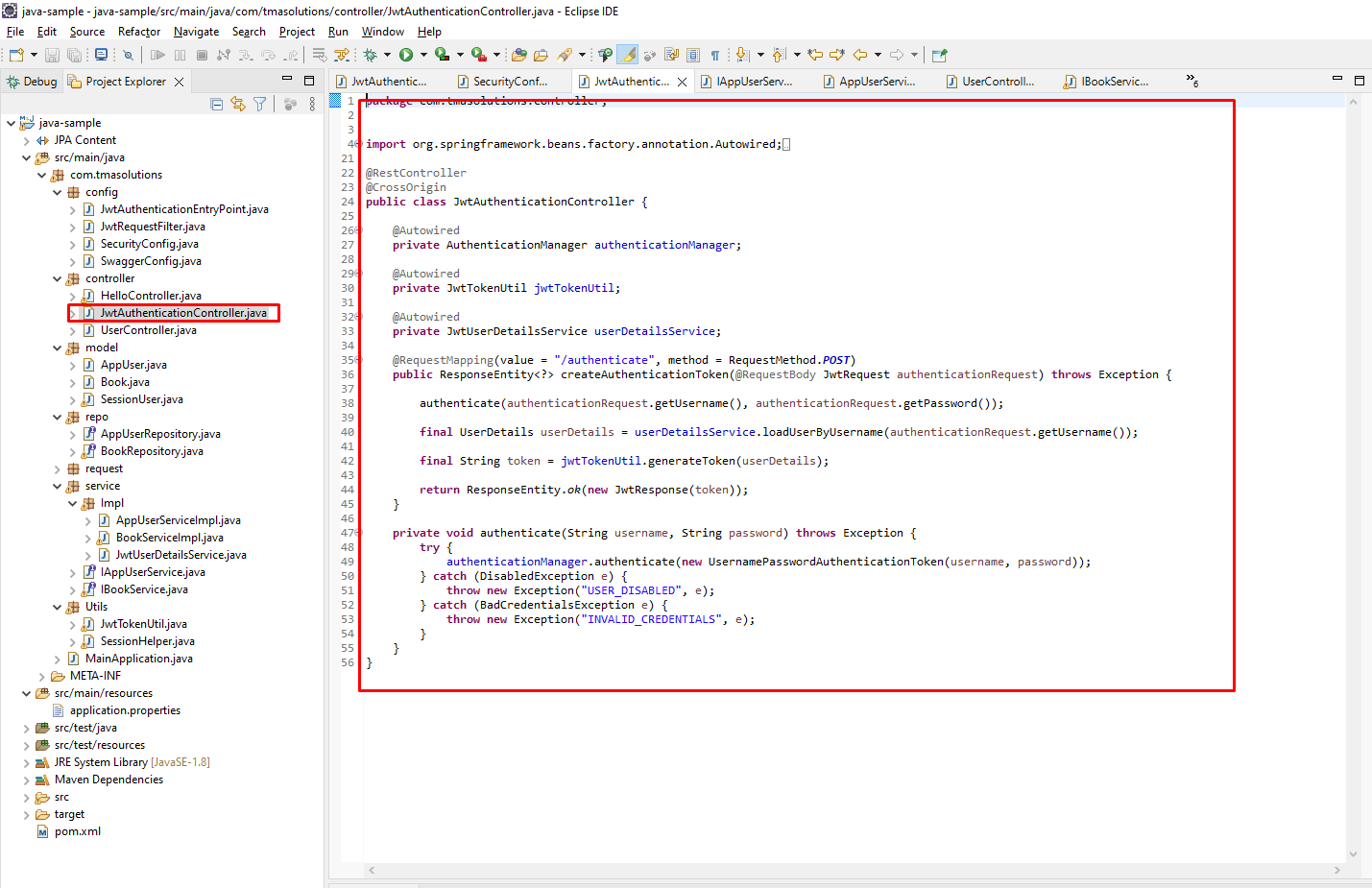
// Add a filter to validate the tokens with every request

httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.**class**);

}

}

1. Create controller “JwtAuthenticationController” to login and return jwt token



**package** com.tmasolutions.controller;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.security.authentication.AuthenticationManager;

**import** org.springframework.security.authentication.BadCredentialsException;

**import** org.springframework.security.authentication.DisabledException;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.web.bind.annotation.CrossOrigin;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

**import** org.springframework.web.bind.annotation.RestController;

**import** com.fasterxml.jackson.core.JsonProcessingException;

**import** com.fasterxml.jackson.databind.JsonMappingException;

**import** com.fasterxml.jackson.databind.JsonNode;

**import** com.fasterxml.jackson.databind.ObjectMapper;

**import** com.tmasolutions.Utils.JwtTokenUtil;

**import** com.tmasolutions.Utils.SessionHelper;

**import** com.tmasolutions.request.JwtRequest;

**import** com.tmasolutions.request.JwtResponse;

**import** com.tmasolutions.service.Impl.JwtUserDetailsService;

@RestController

@CrossOrigin

**public** **class** JwtAuthenticationController {

@Autowired

**private** AuthenticationManager authenticationManager;

@Autowired

**private** JwtTokenUtil jwtTokenUtil;

@Autowired

**private** JwtUserDetailsService userDetailsService;

@RequestMapping(value = "/authenticate", method = RequestMethod.***POST***)

**public** ResponseEntity<?> createAuthenticationToken(@RequestBody JwtRequest authenticationRequest) **throws** JsonMappingException, JsonProcessingException {

**try** {

authenticationManager.authenticate(**new** UsernamePasswordAuthenticationToken(authenticationRequest.getUsername(), authenticationRequest.getPassword()));

**final** UserDetails userDetails = userDetailsService.loadUserByUsername(authenticationRequest.getUsername());

**final** String token = jwtTokenUtil.generateToken(userDetails);

**return** ResponseEntity.*ok*(**new** JwtResponse(token));

}**catch**(Exception ex)

{

ObjectMapper mapper = **new** ObjectMapper();

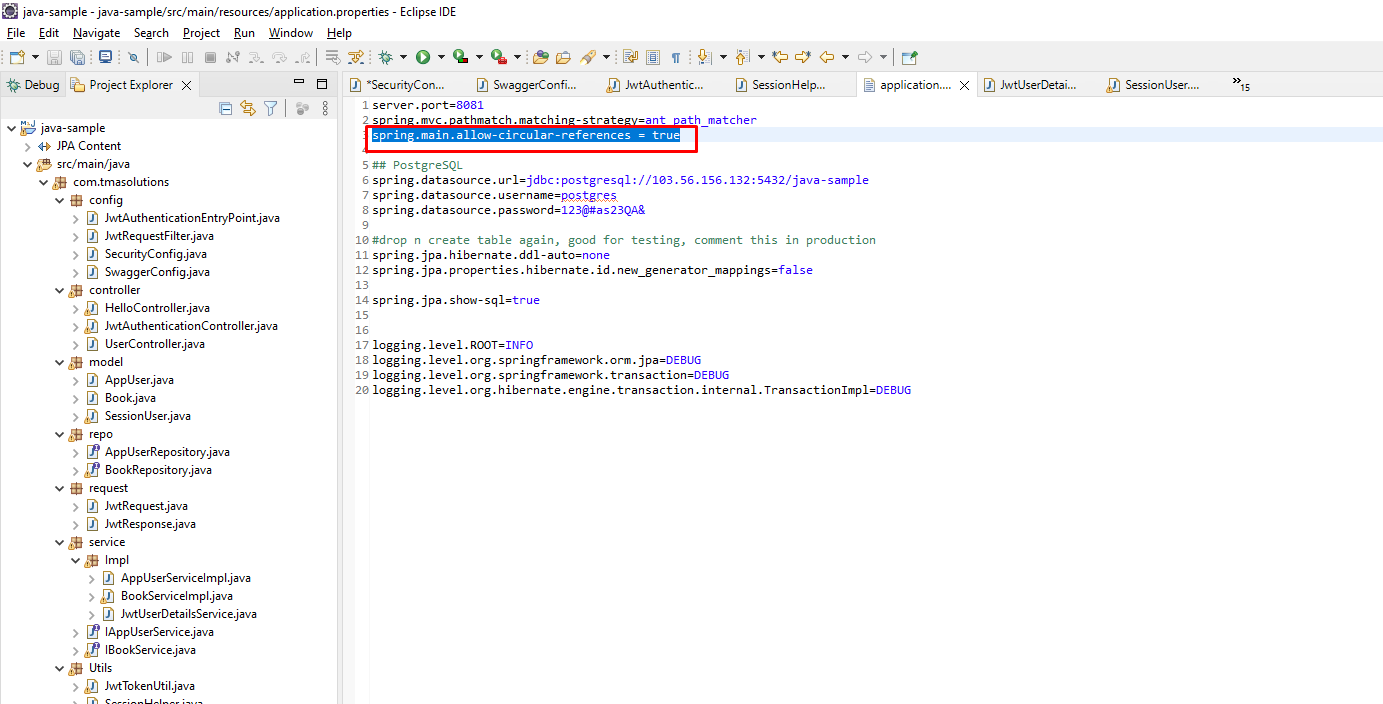
JsonNode json = mapper.readTree("{\"Message\": \"Incorrect User or password\"}");

**return** ResponseEntity.*ok*(json);

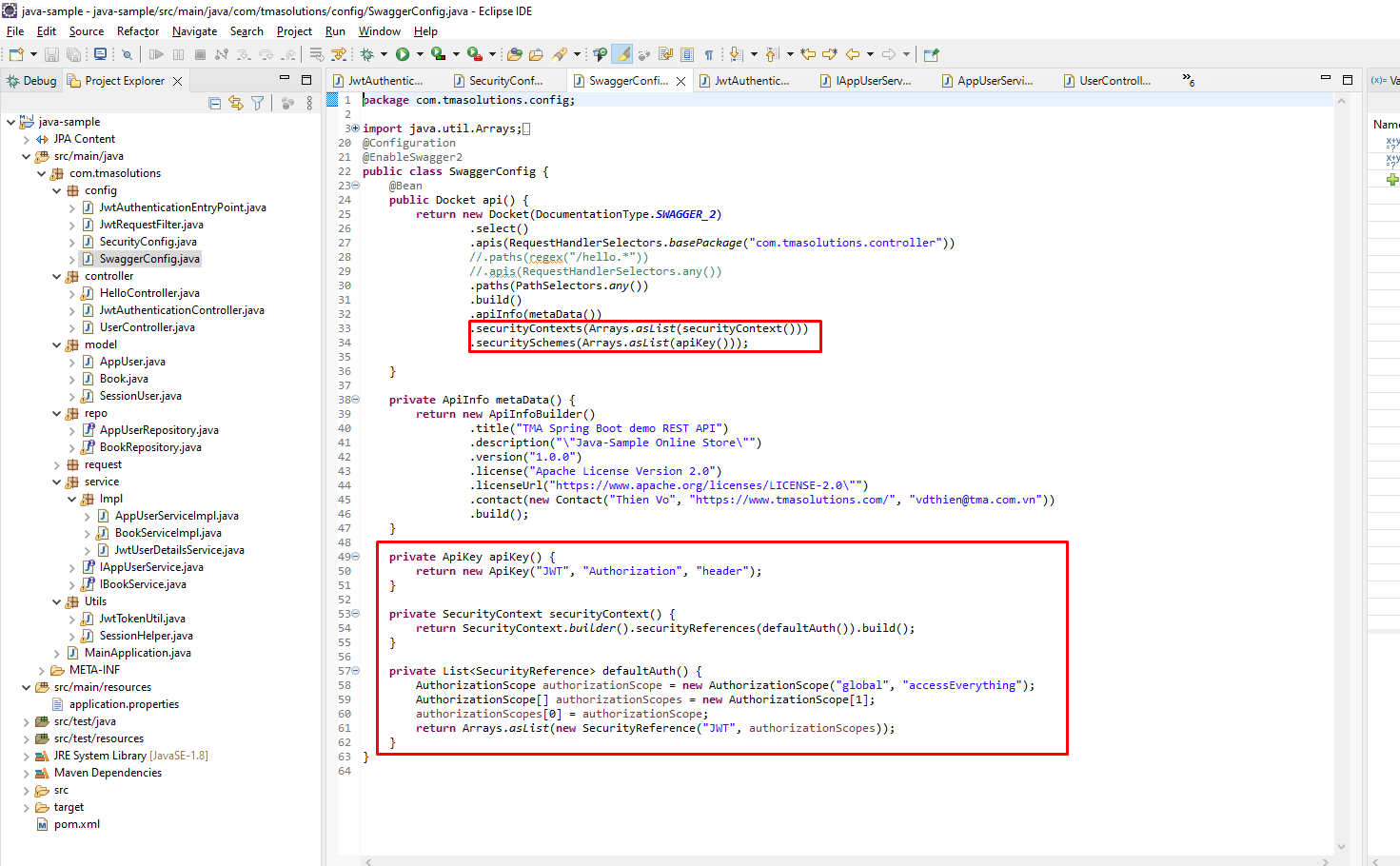
}

}

}



1. **Modify swagger UI to apply authorization to swagger**
2. Modified swagger ui to allow authorize



package com.tmasolutions.config;

import java.util.Arrays;

import java.util.List;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import springfox.documentation.builders.ApiInfoBuilder;

import springfox.documentation.builders.PathSelectors;

import springfox.documentation.builders.RequestHandlerSelectors;

import springfox.documentation.service.ApiInfo;

import springfox.documentation.service.ApiKey;

import springfox.documentation.service.AuthorizationScope;

import springfox.documentation.service.Contact;

import springfox.documentation.service.SecurityReference;

import springfox.documentation.spi.DocumentationType;

import springfox.documentation.spi.service.contexts.SecurityContext;

import springfox.documentation.spring.web.plugins.Docket;

import springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration

@EnableSwagger2

public class SwaggerConfig {

@Bean

public Docket api() {

return new Docket(DocumentationType.SWAGGER\_2)

.select()

.apis(RequestHandlerSelectors.basePackage("com.tmasolutions.controller"))

//.paths(regex("/hello.\*"))

//.apis(RequestHandlerSelectors.any())

.paths(PathSelectors.any())

.build()

.apiInfo(metaData())

.securityContexts(Arrays.asList(securityContext()))

.securitySchemes(Arrays.asList(apiKey()));

}

private ApiInfo metaData() {

return new ApiInfoBuilder()

.title("TMA Spring Boot demo REST API")

.description("\"Java-Sample Online Store\"")

.version("1.0.0")

.license("Apache License Version 2.0")

.licenseUrl("https://www.apache.org/licenses/LICENSE-2.0\"")

.contact(new Contact("Thien Vo", "https://www.tmasolutions.com/", "vdthien@tma.com.vn"))

.build();

}

private ApiKey apiKey() {

return new ApiKey("JWT", "Authorization", "header");

}

private SecurityContext securityContext() {

return SecurityContext.builder().securityReferences(defaultAuth()).build();

}

private List<SecurityReference> defaultAuth() {

AuthorizationScope authorizationScope = new AuthorizationScope("global", "accessEverything");

AuthorizationScope[] authorizationScopes = new AuthorizationScope[1];

authorizationScopes[0] = authorizationScope;

return Arrays.asList(new SecurityReference("JWT", authorizationScopes));

}

}

